

# Using single layer networks for discrete, sequential data: an example from natural language processing

Published in Neural Computing & Applications, 1997, Vol 5 (4)

Caroline Lyon and Ray Frank  
School of Information Sciences  
University of Hertfordshire  
Hatfield, Herts. AL10 9AB

email: C.M.Lyon@herts.ac.uk Tel: +44 (0)1707 284266

## Abstract

Natural Language Processing (NLP) is concerned with processing ordinary, unrestricted text. This work takes a new approach to a traditional NLP task, using neural computing methods. A parser which has been successfully implemented is described. It is a hybrid system, in which neural processors operate within a rule based framework.

The neural processing components belong to the class of Generalized Single Layer Networks (GSLN). In general, supervised, feed-forward networks need more than one layer to process data. However, in some cases data can be pre-processed with a non-linear transformation, and then presented in a linearly separable form for subsequent processing by a single layer net. Such networks offer advantages of functional transparency and operational speed.

For our parser, the initial stage of processing maps linguistic data onto a higher order representation, which can then be analysed by a single layer network. This transformation is supported by information theoretic analysis.

Three different algorithms for the neural component were investigated. Single layer nets can be trained by finding weight adjustments based on (a) factors proportional to the input, as in the Perceptron, (b) factors proportional to the existing weights, and (c) an error minimization method. In our experiments generalization ability varies little; method (b) is used for a prototype parser. This is available via telnet.

**Keywords:** Single layer networks, sequential data, natural language, de-coupled training

## 1 Introduction

This paper examines some of the issues that have to be addressed in designing neural processors for discrete, sequential data. There is a mutual dependence between the representation of data, on the one hand, and the architecture and function of an effective network on the other. As a vehicle for examining these processes we describe an automated partial parser that has been successfully developed [1, 2]. This takes natural language sentences and returns them with the subject and head of the subject located. Ability to generalize is the primary concern. A prototype can be accessed via telnet, on which text can be entered and then parsed. Intermediate steps in the process can be seen.<sup>1</sup>

In principle, simpler networks with well understood functions have *prima facie* advantages, so looking for a representation that enables such networks to be used should be advantageous. With feed forward, supervised networks single layer models enjoy functional transparency and

---

<sup>1</sup>For details, contact author.

operational speed, but in general this type of network will need more than one dynamically linked layer to model non-linear relationships.

However, there is an alternative approach. The layers may be de-coupled, and processing at different layers done in separate steps. Data can be transformed, which is analogous to processing at the first layer, and then presented in a linearly separable form to a single layer net, which is analogous to a second layer. This is illustrated in Figure 4, which shows in simplified form an archetype of the class of Generalized Single Layer Networks (GSLN). A number of different network types that belong to this class are listed by Holden and Rayner [3, page 369]. The critical issue is finding an appropriate non-linear transformation to convert data into the required form.

This paper describes how characteristic linguistic data can be converted into a linearly separable representation that partially captures sequential form. The transformed data is then processed by a single layer network. Three different neural models are tried, and their performance is compared.

All three networks are feed forward models with supervised training. Connection weights can be found by adjustments based on (a) factors proportional to the input (b) factors proportional to the existing weights, and (c) factors related to the difference between desired and actual output, an error minimization method. Model (a) is a traditional Perceptron; model (b) is based on the Hodyne network introduced by Wyard and Nightingale at British Telecom Laboratories [4]; model (c) comes from the class of networks that use an LMS (Least Mean Square error) training algorithm. There is little difference in generalization ability, but network (b) performs slightly better and has been used for the parser in the prototype.

## Natural language processing (NLP)

The automatic parsing of natural language poses a significant problem, and neural computing techniques can contribute to its solution. For an overview of the scope for work in NLP see [5, pages 4-11]. Our prototype gives results of over 90% correct on declarative sentences from technical manuals (see Section 8).

Automated syntactic analysis of natural language has, in the last decade, been characterised by two paradigms. Traditional AI, rule based methods contrast with probabilistic approaches, in which stochastic models are developed from large corpora of real texts. Neural techniques fall into the broad category of the latter, data driven methods, with trainable models developed from examples of known parses. The parser we have implemented uses a hybrid approach: rule based techniques are integrated with neural processors.

Parsing can be taken as a pattern matching task, in which a number of parses are postulated for some text. A classifier distinguishes between the desired parse and incorrect structures. The pattern matching capabilities of neural networks have a particular contribution to make to the process, since they can conveniently model negative as well as positive relationships. The occurrence of some words or groups of words inhibit others from following, and these constraints can be exploited. Arguments on the need for negative information in processing formal languages [6] can be extended to natural language. This is an important source of information which has been difficult for traditional probabilistic methods to access [7, 8]. Neural methods also have the advantage that training is done in advance, so the run time computational load is low.

## Contents of paper

This paper will first take an overview of some factors that are relevant to neural net design decisions, (Section 2). It then looks at characteristics of natural language (Section 3), and the

representation of sequential data (Section 4). A description of the hybrid system used in our work is given, (Section 5). Then we examine some of the design issues for the neural components of this system. First, the data itself is examined closely. Then we consider how the data can be transformed for processing with a single layer net (Section 6). We also comment on the use of a Bayesian classifier, which performs slightly less well than the neural networks. Section 7 describes the three different networks: (a) the Perceptron, (b) Hodyne and (c) an LMS model.

In Section 8 we compare the performance of the three networks. Generalization is good, providing that enough training data is used. Over 90% of the data is correctly classified, and the output can be interpreted so that results for the practical application are up to 100% correct. On the small amount of data processed so far the different networks have roughly comparable generalization ability, but the Hodyne model is slightly better. A discussion on the function of the net follows in Section 9. We conclude (Section 10) that linguistic data is a suitable candidate for processing with this approach.

## 2 Neural nets as classifiers of different types of data

### 2.1 “Clean” and noisy data

Consider the fundamental difference in purpose between systems that handle noisy data, where it is desired to capture an underlying structure and smooth over some input, compared to those that process “clean” data, where every input datum may count. The many applications of neural nets in areas such as image processing provide examples of the first type, the parity problem is typical of the second.<sup>2</sup> These “clean” and “noisy” types can be considered as endpoints of a spectrum, along which different processing tasks lie. In the case of noisy data a classifier will be required to model significant characteristics in order to generalize effectively. The aim is to model the underlying function that generates the data, so the training data should not be over fitted.

On the other hand, for types of data such as inputs to a parity detector, no datum is noise. Consider an input pattern that is markedly different, that is topologically distant, from others in its class. For one type of data this may be noise. In other instances an “atypical” vector may not be noise, and we may need to capture the information it carries to fix the class boundary effectively.

As we demonstrate in the next section, linguistic data needs to be analysed from both angles. We need to capture the statistical information on probable and improbable sequences of words; we also need to use the information from uncommon exemplars, which make up a very large proportion of natural language data.

### 2.2 Preserving topological relationships

Another of the characteristics that is relevant to network design is the extent to which the classification task, the mapping from input to output, preserves topological relationships. In many cases data which are close in input space are likely to produce the same output, and conversely similar classifications are likely to be derived from similar inputs. However, there are other classification problems which are different: a very small change in input may cause a significant change in output and, on the other hand, very different input patterns can belong to

---

<sup>2</sup>The classical parity problem takes a binary input vector, the elements of which are 0 or 1, and classifies it as having an even or odd number of 1’s.

the same class. Again, the parity problem is a paradigm example: in every case changing a single bit in the input pattern changes the desired output.

## 2.3 Data distribution and structure

Underlying data distribution and structure have their effect on the appropriate type of processor, and these characteristics should be examined. Information about the structure of linguistic data can be used to make decisions on suitable representations. In this work information theoretic techniques are used to support decisions on representation of linguistic data.

We may also use information on data distribution to improve generalization ability. As shown in Section 3, assumptions of normality cannot be made for linguistic data. The distribution indicates that in order to generalize adequately the processor must capture information from a significant number of infrequent events.

## 3 Characteristics of linguistic data

The significant characteristics of natural language that we wish to capture include:

- An indefinitely large vocabulary
- The distinctive distribution of words and other linguistic data
- A hierarchical syntactic structure
- Both local and distant dependencies, such as feature agreement

### 3.1 Vocabulary size

Shakespeare is said to have used a vocabulary of 30,000 words, and even an inarticulate computer scientist might need 15,000 to get by (counting different forms of the same word stem separately). Current vocabularies for commercial speech processing databases are  $O(10^5)$ . Without specifying an upper limit we wish to be able to model an indefinite number of words.

### 3.2 The distinctive distribution of linguistic data

The distribution of words in English and other languages has distinctive characteristics, to which Shannon drew attention [9]. Statistical studies were made of word frequencies in English language texts. In about 250,000 words of text (word tokens) there were about 30,000 different words (word types), and the frequency of occurrence of each word type was recorded. If word types are ranked in order of frequency, and  $n$  denotes rank, then there is an empirical relationship between the probability of the word at rank  $n$  occurring,  $p(n)$ , and  $n$  itself, known as Zipf's Law:

$$p(n) * n = \text{constant}$$

This gives a surprisingly good approximation to word probabilities in English and other languages, and indicates the extent to which a significant number of words occur infrequently. For example, words that have a frequency of less than 1 in 50,000 make up about 20-30% of typical English language news-wire reports [10]. The LOB corpus<sup>3</sup> with about 1 million word

---

<sup>3</sup>The London Oslo Bergen corpus is a collection of texts used as raw material for natural language processing

tokens contains about 50,000 different word types, of which about 42,000 occur less than 10 times each [11].

The “zipfian” distribution of words has been found typical of other linguistic data. It is found again in the data derived from part-of-speech tags used to train the prototype described here: see Figures 2 and 3.

Other fields in which zipfian distribution is noted include information retrieval and data mining (e.g. characteristics of WWW use, patterns of database queries). It has also been observed in molecular biology (e.g. statistical characteristics of RNA landscapes, DNA sequence coding).

### Mapping words onto part-of-speech tags

In order to address the problem of sparse data the vocabulary can be partitioned into groups, based on a similarity criterion, as is done in our system. An indefinitely large vocabulary is mapped onto a limited number of part-of-speech tag classes. This also make syntactic patterns more pronounced. Devising optimal tagsets is a significant task, on which further work remains to be done. For the purpose of this paper we take as given the tagsets used in the demonstration prototype, described in [12]. At the stage of processing described in this paper 19 tags are used.

### 3.3 Grammatical structure

There is an underlying hierarchical structure to all natural languages, a phenomenon that has been extensively explored. Sentences will usually conform to certain structural patterns, as is shown in a simplified form in Figure 1. This is not inconsistent with the fact that acceptable grammatical forms evolve with time, and that people do not always express themselves grammatically. Text also, of course, contains non-sentential elements such as headings, captions, tables of contents. The work described in this paper is restricted to declarative sentences.

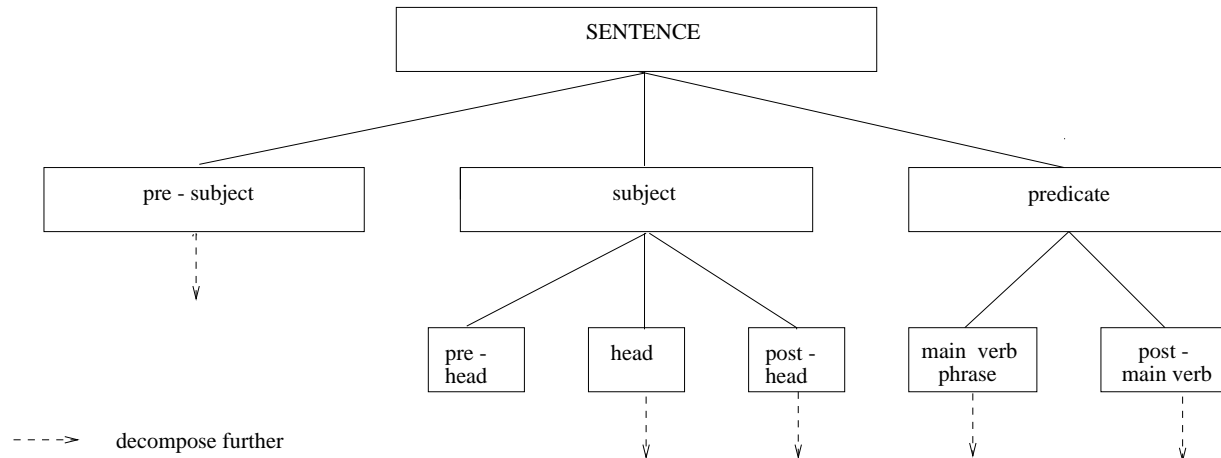


Figure 1: Decomposition of the sentence into syntactic constituents.

Within the grammatical structure there is an indefinite amount of variation in the way in which words can be assembled. On the other hand, the absence or presence of a single word can make a sentence unacceptable, for example

- \*There are many problems arise. ... (1)
- \*We is late. ... (2)

Now consider how linguistic data fits into the scheme described in Section 2.2 on preserving topological relationships. Many strings of words that are close in input space are also in the same grammatical category, but, conversely, on occasions a single word change can put a string into a different category. Our processor has to model this.

### 3.4 Local and distant dependencies

In examining natural language we find there are dependencies between certain words, both locally and at a distance. For instance, a plural subject must have a plural verb, so a construction like sentence (2) above is incorrect. This type of dependency in its general form is not necessarily local. In sentence (3) below the number of the subject is determined by its head, which is not adjacent to the verb:

- If a cooler is fitted to the gearbox, [ the pipe [ connections ] of the cooler ] must be regularly checked for corrosion. ... (3)

The subject of this sentence is the plural “connections”. Note that modal verbs like “must” have the same singular and plural form in English, but not in many other languages. For an automated translation system to process modal verbs it is necessary to find the head of the subject that governs the verb and ensure number agreement.

There are also dependencies between sentences and between more distant parts of a text. We aim to model just the intra-sentential dependencies as our automatic parser is developed.

## 4 Modelling sequential data

Three methods have commonly been used to model sequential data, such as language, for connectionist processing. The first is to move a window along through the sequence, and process a series of static “snapshots”. Within each window ordering information is not represented. Sejnowski’s NETtalk is a well known example [13].

Another method that warrants further investigation is the use of recurrent nets [14, 15]. In its basic form this type of network is equivalent to a finite state automaton that can model regular languages [16].

### 4.1 The n-gram method

The third method, used in this work, is to take sets of ordered, adjacent elements, which capture some of the sequential structure of language. This is related to the well known trigram approach used in probabilistic language processing. Combining tags into higher order tuples can also act as a pre-processing function, making it more likely that the transformed data can be processed by a single layer network (Section 6).

This method of representation captures some of the structure of natural language, as is shown by analysis with information theoretic techniques. There are relationships between neighbouring words in text: some are likely to be found adjacent, others are unlikely. When words are mapped onto part-of-speech tags this is also the case. This observation is supported by an investigation of entropy levels in the LOB corpus, in which 1 million words have been manually tagged.

Entropy can be understood as a measure of uncertainty [17, chapter 2]. The uncertainty about how a partial sequence will continue can be reduced when statistical constraints of neighbouring elements are taken into account. Shannon introduced this approach by analysing sequences of letters [9], where the elements of a sequence are single letters, adjacent pairs or triples, with order preserved. These are  $n$ -grams, with  $n$  equal to 1, 2 or 3. The entropy of a sequence represented by letter  $n$ -grams declines as  $n$  increases. When sequences of tags in the LOB corpus were analysed the same result was obtained: the entropy of part-of-speech  $n$ -grams declines as  $n$  increases from 1 to 3. This indicates that some of the structure of language is captured by taking tag pairs and triples as processing elements.

We adopt the common approach of presenting data as binary vectors for all the networks examined in this work. Each element of the input vector represents an ordered tuple of adjacent part-of-speech tags, a pair or a triple. If a given tag tuple is present in an input string, then that element in the input vector is flagged to 1, else it remains 0.

## 5 Description of the hybrid natural language processor

In order to process unrestricted natural language it is necessary to attack the problem on a broad front, and use every possible source of information. In our work the neural networks are part of a larger system, integrated with rule based modules. We first *assert* that there is a syntactic structure which can be mapped onto a sentence (Figure 1). Then we use neural methods to find the mapping in each particular case. The grammar used is defined in [12, chapter 5].

### 5.1 Problem decomposition

In order to effect the mapping of this structure onto actual sentences we decompose the problem into stages, finding the boundaries of one syntactic feature at a time. The first step is to find the correct placement for the boundaries of the subject, then further features are found in the 3 basic constituents. In the current prototype the head of the subject is subsequently identified. The processing at each stage is based on similar concepts, and to explain the role of the neural networks we shall in this paper discuss the first step in which the subject is found.

The underlying principle employed each time is to take a sentence, or part of a sentence, and generate strings with the boundary markers of the syntactic constituent in question placed in all possible positions. Then a neural net selects the string with the correct placement. This is the grammatical, “yes” string for the sentence. The model is trained in supervised mode on marked up text to find this correct placement. The different networks that are examined share the same input and output routines, and each was integrated into the same overall system.

### 5.2 Tagging

The first stage in both the training and testing process is to map an indefinite number of words onto a limited number of part-of-speech tags. An automatic tagger allocates one or more part-of-speech tags to the words to be processed. Many words, typically perhaps 25% to 30%, have more than one tag. The CLAWS automatic tagger [18] provided a set of candidate tags for each word, but the probabilistic disambiguation modules were not used: disambiguating the tags is a sub-task for the neural processor. The CLAWS tagset was mapped onto a customised tagset of 19, used for the work described here. Further information on tagset development is in [12, chapter 4], and, briefly, in [1].

### 5.3 Hypertags as boundary markers

As well as part of speech tags we also introduce the syntactic markers, virtual tags, which at this stage of the process will demarcate the subject boundary. These *hypertags* represent the opening ‘[’ and closing ‘]’ of the subject. The hypertags have relationships with their neighbours in the same way that ordinary tags do: some combinations are likely, some are unlikely. The purpose of the parser is to find the correct location of the hypertags.

With a tagset of 19 parts-of-speech, a start symbol and 2 hypertags we have 22 tags in all. Thus, there are potentially  $22^2 + 22^3 = 11132$  pairs and triples. In practice only a small proportion of tuples are actually realised - see Tables 3 and 4 . At other stages of the parsing process larger tagsets are required (see [12]).

### 5.4 Rule based pruning: the Prohibition Table

Strings can potentially be generated with the hypertags in all possible positions, in all possible sequences of ambiguous tags. However, this process would produce an unmanageable amount of data, so it is pruned by rule based methods integrated into the generation process. Applying local and semi-local constraints the generation of any string is zapped if a prohibited feature is produced. For fuller details see [12] or [1]. An example of a local prohibition is that the adjacent pair (**verb, verb**) is not allowed. Of course (**auxiliary verb, verb**) is permissible, as is (**verb, ] , verb**). These rules are similar to those in a constraint grammar, but are not expected to be comprehensive. There are also arbitrary length restrictions on the sentence constituents: currently, the maximum length of the pre-subject is 15 words, of the subject 12 words.

### 5.5 Neural processing

This pruning operation is powerful and effective, but it still leaves a set of candidate strings for each sentence - typically between 1 and 25 for the technical manuals. Around 25% of sentences are left with a single string, but the rest can only be parsed using the neural selector. This averages at about 3 for the technical manuals, more for sentences from other domains. In training we manually identify the string with the correct placement of hypertags, and the correctly disambiguated part-of-speech tags. In testing mode, the correct string is selected automatically.

## 5.6 Coding the input

As an example of input coding consider a short sentence:

All papers published in this journal are protected by copyright. ....(4)

(A) Map each word onto 1 or more tags

all	predeterminer
papers	noun or verb
published	past-part-verb
in	preposition or adverb
this	pronomial determiner
journal	noun
are	auxiliary-verb
protected	past-part-verb
by	preposition
copyright	noun
.	endpoint

(B) Generate strings with possible placement of subject boundary markers,  
and possible tag allocations (pruned).

string no. 1

strt [ pred ] verb pastp prep prod noun aux pastp prep noun end

.....

string no. 4

strt [ pred noun ] pastp adv prod noun aux pastp prep noun end

string no. 5

strt [ pred noun pastp ] adv prod noun aux pastp prep noun end

string no. 6 \*\*\* target \*\*\*

strt [ pred noun pastp prep prod noun ] aux pastp prep noun end

string no. 7

strt [ pred noun pastp adv prod noun ] aux pastp prep noun end

(C) Transform strings into sets of tuples

string no. 1

(strt, [ ] ( [ , pred ) ( pred, ] ) .....(noun, end)

(strt, [ , pred) ([ , pred, ] ) (pred, ] , verb)..... (prep, noun, end)

and similarly for other strings

(D) The elements of the binary input vector represent all tuples, initialized to 0. If a tuple is present in a string the element that represents it is changed from 0 to 1.

## 5.7 Characteristics of the data

The domain for which our parser was developed was text from technical manuals from Perkins Engines Ltd. They were written with the explicit aim of being clear and straight forward [19]. Using this text as a basis we augmented it slightly to develop the prototype on which users try their own text. Declarative sentences were taken unaltered from the manuals for processing: imperative sentences, titles, captions for figures were omitted. 2% of declarative sentences were omitted, as they fell outside the current bounds (e.g. the subject had more than 12 words). A corpus of 351 sentences was produced: see Table 1.

Number of sentences	351
Average length	17.98 words
No. of subordinate clauses:	
In pre-subject	65
In subject	19
In predicate	136
Co-ordinated clauses	50

Table 1: Corpus statistics. Punctuation marks are counted as words, formulae as 1 word.

This corpus (Tr-all) was divided up 4 ways (Tr 1 to Tr 4) so that nets could be trained on part of the corpus and tested on the rest, as shown in Table 2. In order to find the placement of the subject boundary markers we do not need to analyse the predicate fully, so the part of the sentence being processed is dynamically truncated 3 words beyond the end of any postulated closing hypertag. The pairs and triples generated represent part of the sentence only.

## 5.8 Data distribution

Statistics on the data generated by the Perkins corpus are given in Tables 3, 4 and 5. A significant number of tuples occur in the test set, but have not occurred in the training set, since, as Figures 2 and 3 show, the distribution of data has a zipfian character.

Training set	number of sentences	number of strings	Test set	number of sentences	number of strings	Ratio of testing/training strings
Tr-all	351	1037				
Tr 1	309	852	Ts 1	42	85	0.10
Tr 2	292	863	Ts 2	59	174	0.20
Tr 3	288	843	Ts 3	63	194	0.23
Tr 4	284	825	Ts 4	67	212	0.26

Table 2: Description of training and test sets of data

Training set	number of pairs in ‘yes’ strings	number of pairs in ‘no’ strings	Test set	number of new pairs in ‘yes’ strings	number of new pairs in ‘no’ strings
Tr-all	162	213			
Tr 1	161	211	Ts 1	1 (1%)	2 (1%)
Tr 2	160	210	Ts 2	2 (1%)	3 (1%)
Tr 3	156	210	Ts 3	6 (4%)	3 (1%)
Tr 4	149	193	Ts 4	13 (9%)	20 (10%)

Table 3: Part-of-speech pairs in training and testing sets. ‘Yes’ indicates correct strings, ‘no’ incorrect ones

Training set	number of triples in ‘yes’ strings	number of triples in ‘no’ strings	Test set	number of new triples in ‘yes’ strings	number of new triples in ‘no’ strings
Tr-all	406	727			
Tr 1	400	713	Ts 1	6 (2%)	14 (2%)
Tr 2	383	686	Ts 2	23 (6%)	41 (6%)
Tr 3	361	642	Ts 3	45 (12%)	85 (13%)
Tr 4	364	632	Ts 4	42 (12%)	95 (15%)

Table 4: Part-of-speech triples in training and testing data sets

Total number of pairs in ‘yes’ strings	4142
Total number of pairs in ‘no’ strings	8652
Total number of triples in ‘yes’ strings	3736
Total number of triples in ‘no’ strings	8021

Table 5: Total number of tuples in Tr-all, including repetitions

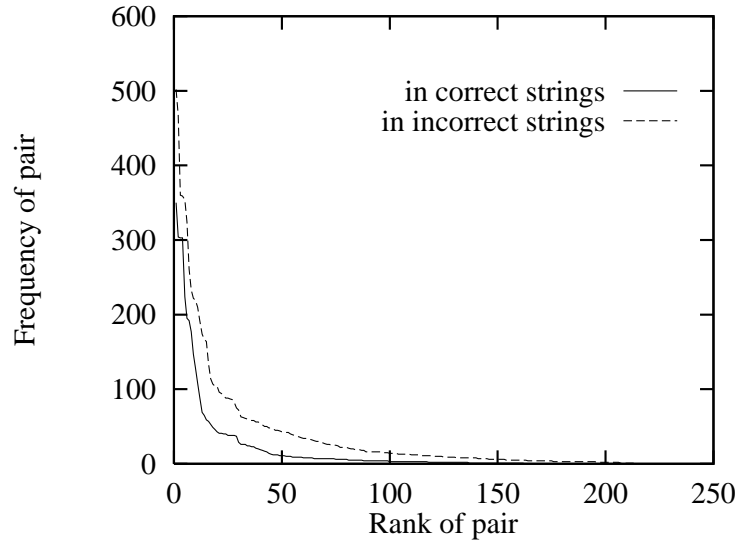


Figure 2: Data from 351 sentences in technical manuals. Pairs are ranked by frequency of occurrence in correct and incorrect strings. Relationship between rank and frequency shown.

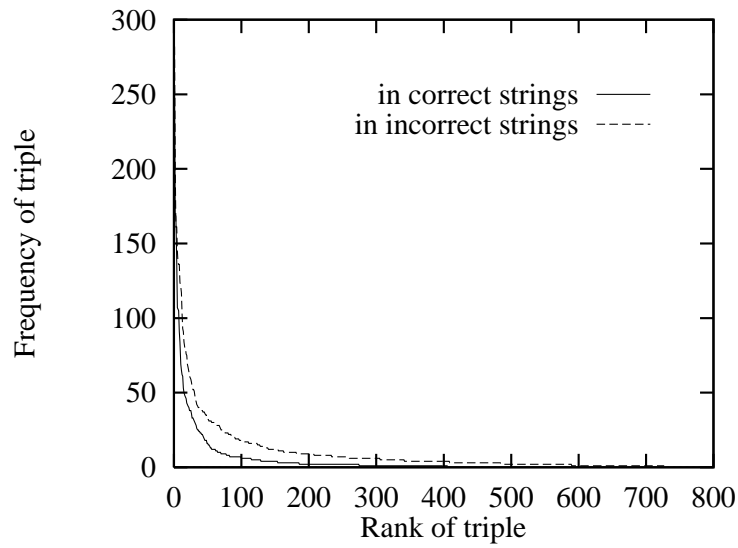


Figure 3: Relationship between rank and frequency of triples on the same data.

## 5.9 Interpreting the output

For training, the set of strings generated by the training text is taken as a whole. Each string is given a “grammaticality measure”  $\Gamma$ , where  $\Gamma$  is positive for a correct string, negative for an incorrect one. Details follow in Section 7. In testing mode we consider separately the set of strings generated for each sentence, and the string with the highest  $\Gamma$  measure is taken as the correct one for that sentence. In the example in Section 5.6 string 6 is the correct one.

However, on test data, there are 4 metrics for correctness that can be useful in different practical situations. The measure “correct-a” requires only that the hypertags are correctly placed: string 7 as well as string 6 is correct-a. “correct-b” requires also that all words within the subject are correctly tagged, “correct-c” that all words within the part of the sentence being processed are correctly tagged. The final measure “correct-d” records the proportion of strings that are in the right class. It can happen that the highest scoring string may have a negative  $\Gamma$ . Conversely, some incorrect strings can have a positive  $\Gamma$  without having the highest score. For practical purposes the measures “correct-a”, “-b”, and “-c” will be the significant ones. But in analysing the performance of the networks we will be interested in “correct-d”, the extent to which the net can generalize and correctly classify strings generated by the test data.

## Metrics of other systems

Note that these measures relate to a string, not to individual elements of the string. This contrasts with some natural language processing systems, in which the measure of correctness relates to each word. For instance, automated word tagging systems typically measure success by the proportion of *words* correctly tagged. The best stochastic taggers typically quote success rates of 95% to 97% correct. If sentences are very approximately 20 words long on average, this can mean that there is an error in many sentences.

## 6 Using single layer networks

### 6.1 Conversion to linearly separable forms

It is always theoretically possible to solve supervised learning problems with a single layer, feed forward network, providing the input data is enhanced in an appropriate way. A good explanation is given by Pao [20, chapter 8]. Whether this is desirable in any particular case must be investigated. The enhancement can map the input data onto a space, usually of higher dimensionality, where it will be linearly separable. Widrow’s valuable 1990 paper on “Perceptron, Madaline, and Back Propagation” [21, page 1420] explores these approaches “which offer great simplicity and beauty”.

Figure 4 illustrates the form of the Generalized Single Layer Network (GSLN). This figure is derived from Holden and Rayner [3]. A non-linear transformation  $\Phi$  on inputs  $\{x\}$  converts them to elements  $\{y\}$ , which a single layer net will then process to produce an output  $z$  (temporarily assuming 1 output). The  $\Phi$  functions, or basis functions, can take various forms. They can be applied to each input node separately, or, as indicated in the figure, they can model the higher order effects of correlation. In our processor  $\Phi$  is an ordered ‘AND’, described in the following section. A similar function is used in the grammatical inference work of Giles et al. [15]; it is also used in DNA sequence analysis [22]. The  $\Phi$  function can be arithmetic: for instance, for polynomial discriminant functions the elements of the input vectors are combined as products [23, page 135]. Successful uses of this approach include the discrimination of different vowel sounds [24] and the automated interpretation of telephone company data in tabular form [25].

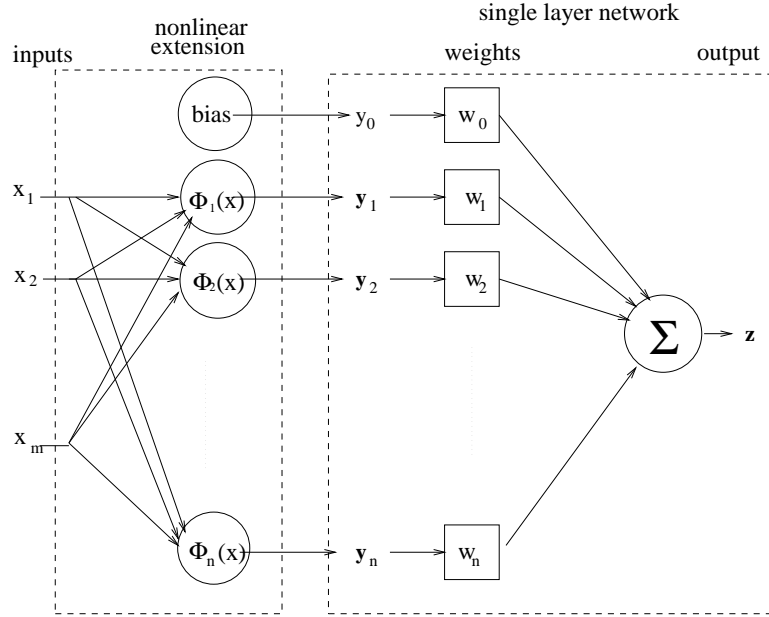


Figure 4: The Generalized Single Layer Network, GSLN, with 1 output

Radial basis function (RBF) networks also come into the class of GSLNs. In their two stage training procedure the parameters governing the basis functions are determined first. Then a single layer net is used for the second stage of processing. Examples include Tarassenko's work on the analysis of ECG signals [26] and a solution to an inverse scattering problem to determine particle properties [27].

An important characteristic of the GSLN is that processing at different layers is de-coupled. The first stage of training is unsupervised: the  $\Phi$  functions are applied without recourse to desired results. In the second stage of training a supervised method is used, in which weight adjustments on links are related to target outputs.

One perspective on the GSLN is given by Bishop [28, e.g. page 89], who characterises this type of system as a special case of the more general multi-layer network. Whereas in the general case the basis functions at the first layer are modified during the training process, in this type of system the basis functions are fixed independently. Widrow [21, page 1420] puts it the other way round: "one can view multi-layer networks as single layer networks with trainable preprocessors...".

## 6.2 The conversion function used in the parser

We use this approach in the parser by converting a sequence of tags to a higher order set of adjacent pairs and triples. The example in section 5.6, stage C shows how the input elements are constructed. Thus, one of the elements derived from string 1 is ( **[, predeterminer, verb** ). This of course is not the same as ( **predeterminer, [, verb** ). The same tag can be repeated within a tuple: *Computer Science Department* maps onto ( **noun, noun, noun** ).

This can be related to Figure 4. Let each  $x_i$  for  $i = 1$  to  $m$  represent a tag. The  $\Phi$  functions map these onto  $m^2$  pairs and  $m^3$  triples, so  $n = m^2 + m^3$ . If  $s$  is a sequence of  $l$  tags, it is transformed into a set  $S$  of higher order elements by  $\Phi_p$  for pairs and  $\Phi_t$  for triples.

$$s = x_1 \dots x_i, x_{i+1}, x_{i+2} \dots x_l$$

$$\Phi_p(x_i) = (x_i, x_{i+1}) \quad \text{for } i = 1 \text{ to } i = l - 1$$

$$\Phi_t(x_i) = (x_i, x_{i+1}, x_{i+2}) \quad \text{for } i = 1 \text{ to } i = l - 2$$

$$S = \{\Phi_p(x_i)\} \cup \{\Phi_t(x_i)\}$$

For some of our investigations either pairs or triples were used, rather than both.

The  $\Phi$  function represents an ordered ‘AND’: the higher order elements preserve sequential order. This function was derived using heuristic methods, but the approach was supported by an objective analysis of the proposed representation. We aimed to capture *some* of the implicit information in the data, model invariances, represent structure. As described in Section 4.1, the choice of the tupling pre-processing function is supported by information theoretic analysis. It captures local, though not distant, dependencies. Using this representation we address simultaneously the issues of converting data to a linearly separable form, modelling its sequential character and capturing some of its structure.

An approach similar to our own has been used to develop neural processors for an analysis of DNA sequences [22, page 166]. Initially a multi-layer network was used for one task, but an analysis of its operation led to the adoption of an improved representation with a simpler network. The input representing bases was converted to codons (tuples of adjacent bases), and then processed with a Perceptron.

## 6.3 The practical approach

Minsky and Papert acknowledged that single layer networks could classify linearly inseparable data if it was transformed to a sufficiently high order [29, page 56], but claimed this would be impractical. They illustrated the point with the example of a parity tester. However, this example is the extreme case, where *any* change of a single input element will lead to a change in the output class. If there are  $n$  inputs, then it is necessary to tuple each element together to  $O(n)$ . The consequent explosion of input data would make the method unusable for all but the smallest data sets.

However, in practice, real world data may be different. Shavlik et al [30] compare single and multi-layer nets on 6 well known problems, and conclude “Regardless of the reason, data for many ‘real’ problems seems to consist of linearly separable categories..... Using a Perceptron as an initial test system is probably a good idea.” This empirical approach is advocated here. Tests

for linear separability and related problems are computationally heavy [31, 32], so we tried single layer networks to see whether the higher order data we use is in practice linearly separable, or nearly so.

Taking data items as pairs typically produces training sets of which about 97% can be learnt by a single layer network; taking triples raises learnability to about 99%. Thus our data is almost linearly separable.

## 6.4 Linear discriminants - neural and Bayesian methods

Having established empirically that after transformation we have a linear problem, there are a number of different methods of linear discriminant analysis that could be used. Our single layer networks are convenient tools.

We also ran our data through a Bayesian classifier, based on the model described by Duda and Hart [23, page 32]. Results were about 5% less good on test data than those from Hodyne. Though the parsing problem is decomposed so that good estimates can usually be made of prior probabilities, estimating class conditional probabilities needs further investigation. (If  $n$  possible parses are generated and 1 is correct, then the prior is  $1/n$ ). Frequency counts extracted from the training data cannot be used as they stand as probability estimates. The zipfian distribution of data can distort the probabilities, even when very large quantities are used, so that rare events are given too much significance. Moreover, further information on zero frequency items, though limited, can be extracted using an appropriate technique, as Dunning shows [10]. There are a number of methods of estimating probabilities on the basis of partial information which need investigation [33, page 55].

These issues can be avoided by using neural discriminators.

## 6.5 Training set size

There is a relationship between training set size and linear separability. Cover’s classical work addressed the probability that a set of *random*, real valued vectors with random binary desired responses are linearly separable [34, 21]. Using his terminology and taking the term “pattern” to mean a training example, the critical factor is the ratio of number of patterns,  $\Pi$ , to number of elements in each pattern,  $n$ . While  $\Pi/n < 1$ , the probability  $P_{separable} = 1.0$ . If  $\Pi/n = 1$  then  $P_{separable} = 0.5$ . As  $\Pi/n$  increases,  $\Pi_{separable}$  quickly declines.

These observations are given as background information to indicate that training set size should be considered, but they do not apply in our case as they stand. First, our data is not random. Secondly, a necessary condition that the vectors are in “general position”, normally satisfied by real valued vectors, may not hold for binary vectors [35, page 97].

The number of training examples,  $\Pi$ , is a factor in determining generalization capability (see, for example, [36, 3]). The probability that an error is within a certain bound increases with the number of training examples. Decreasing  $\Pi$  to convert data to a linearly separable form would be profitless.

The ratio of training examples to weights in our data is shown in Figure 5. Note that the corpus used for this preliminary working prototype is small compared to other corpora, and future work will use much larger ones, which could affect this ratio.

## 7 Three single layer networks

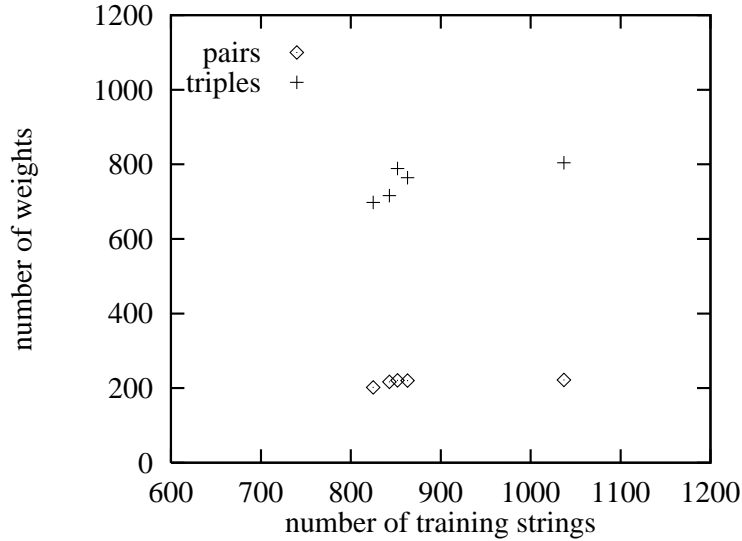


Figure 5: Relationship between number of training examples and number of weights, for the Perceptron and LMS nets with one output.

## 7.1 Architecture

Refer again to Figure 4, illustrating the GSLN. In this work we compare 3 networks, which all use the same  $\Phi$  functions, described in Section 6.1. We now compare methods of processing at the second stage, that is the performance of 3 different single layer classifiers. The Perceptron and LMS net can be characterised as examples of the GSLN in Figure 4. The Hodyne model differs in having 2 outputs, not symmetrically connected, as in Figure 6.

## 7.2 Methods of adjusting connection weights during training

When single layer networks are used, we do not have the classic problem of “credit assignment” associated with multi-layer networks: the input neurons responsible for incorrect output can be identified, and the weights on their links adjusted. There is a choice of methods for updating weights. We do not *have* to use differentiable activation functions, as in the multi-layer Perceptron. These methods can therefore be divided into two broad categories. First there are “direct update” methods, used in the traditional Perceptron and Hodyne-type nets, where a weight update is only invoked if a training vector falls into the wrong class. This approach is related to the ideas behind reinforcement learning, but there is no positive reinforcement. If the classification is correct weights are left alone. If the classification is incorrect then the weights are incremented or decremented. No error measure is needed: the weight update is a function either of the input vector (Perceptron), or of the existing weights (Hodyne).

Secondly, there are “error minimization” approaches, which can also be used in multi-layer nets. An error measure, based on the difference between a target value and the actual value of the output, is used. This is frequently, as in standard back propagation, a process based on minimizing the mean square error, to reach the LMS error [37]. We have used a modified error minimization method (Section 7.5).

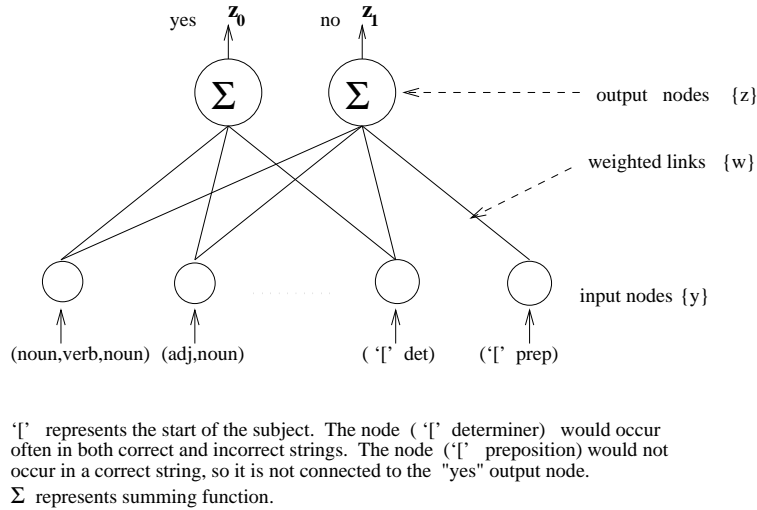


Figure 6: The Hodyne network.

### 7.3 The Perceptron

The Perceptron and LMS models are both fully connected nets with a single output. Details of the well known Perceptron training algorithm, and of the parameters used, are in [38]. The output represents  $\Gamma$ , the grammaticality measure of the input string. In training, a grammatical string must produce a positive output, an ungrammatical string a negative output. The wrong output triggers a weight adjustment on all links contributing to the result. This is a function of the normalized input values, scaled by the learning rate.

To speed the training process, a method of “guided initialization” sets initial random weight within bounds determined by the expected output. To implement this, see whether a new, previously unseen input element belongs to a “yes” or “no” string, corresponding to desired positive or negative outputs. Then set a random value between 0.0 and 0.3 for “yes”, between 0.0 and  $-0.3$  for “no” [12]. When training is finished, weights on links from unvisited input elements are set to 0.0.

Recall that in testing mode we consider the set of strings generated for each sentence. and the string with the highest  $\Gamma$  measure is taken as the correct one for that sentence.

### 7.4 Hodyne

This network, shown in Figure 6, is derived from the model introduced by Wyard and Nightingale [4]. The 2 outputs  $z_0$  and  $z_1$  represent grammatical and ungrammatical, “yes” and “no”, results. In training, a grammatical string must produce  $z_0 > z_1$ , and vice-versa, else a weight adjustment is invoked. In testing mode, as for the Perceptron, the strings generated by each sentence are processed, and the string with the highest  $\Gamma$  score for that sentence is the winner. For this network the grammaticality measure  $\Gamma$  is  $z_0 - z_1$ . Since it is not widely known a summary of the training method follows. More implementation details can be found in [12].

### Notation

Let each input vector have  $n$  elements  $y_i$ . Let  $w(t)_{i,j}$  be the weight from the  $i$ th input node to the  $j$ th output node at time  $t$ . Let  $u(t)_{i,j}$  be the update factor.

$\delta = -1$  or  $\delta = +1$  indicates whether weights should be decremented or incremented.

Mark all links disabled.

Initially, percentage of strings correctly classified = 0.0

REPEAT

from START1 to END1 until % strings correctly classified exceeds chosen threshold:

START1

REPEAT from START2 to END2 for each string

START2

Present input, a binary vector,  $y_1, y_2, \dots, y_n$

Present desired output,  $z_0 > z_1$  or *vice versa*

For any  $y > 0$  enable link to desired result if it is disabled

Initialize weight on any new link to 1.0

Calculate outputs  $z_0$  and  $z_1$

$$z_k = \sum_{i=1}^{i=n} w_{i,k} * y_i$$

If actual result = desired result

Count string correct, leave weights alone

Else adjust weights on current active links:

if  $z_0 < z_1$  then  $\delta = +1$  on links to  $z_0$ ,  $\delta = -1$  on links to  $z_1$   
and *vice versa*

$$w(t+1)_{i,j} = \left[ 1 + \frac{\delta * w(t)_{i,j}}{1 + (\delta * w(t)_{i,j})^4} \right] w(t)_{i,j}$$

END2

Calculate % strings correctly classified. If greater than threshold, terminate

END1

For the Hodyne type net the update factor  $u$  is a function of the current weights; as the weights increase it is asymptotic to 0, as they decrease it becomes equal to 0.

$$u(t)_{i,j} = \frac{\pm w(t)_{i,j}^2}{1 \pm w(t)_{i,j}^4}$$

This function satisfies the requirement that the weights increase monotonically and saturate (see Figure 7). We use the original Hodyne function with a comparatively low computational load. Note that in contrast to the Perceptron, where the learning rate is set at compile time, the effective learning rate in this method varies dynamically. The greatest changes occur when weights are near their initial values of 1.0, as they get larger or smaller the weight change decreases.

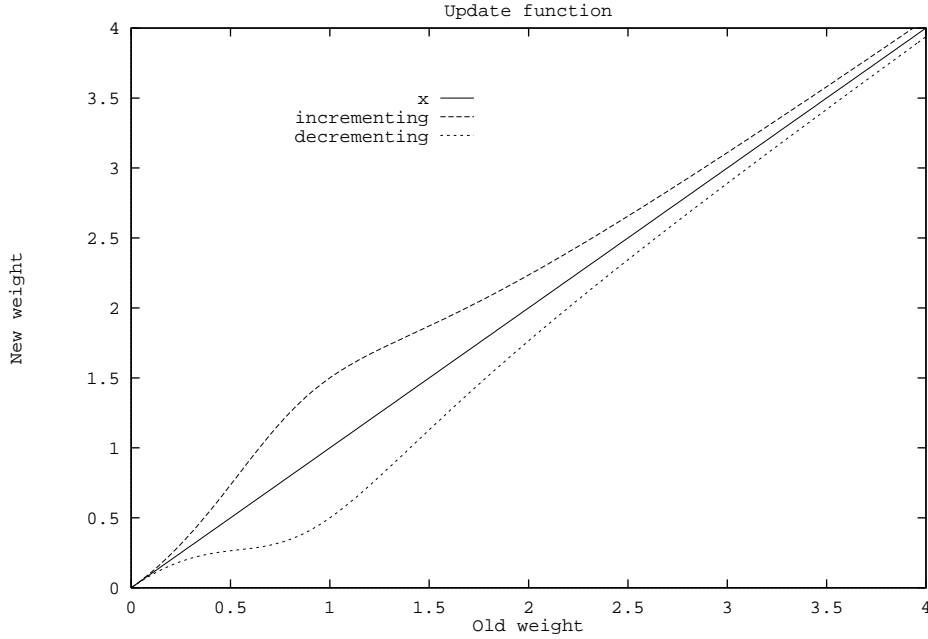


Figure 7: Relationship between old and new weights for the Hodyne net

### Hodyne’s pattern of connectivity and the Prohibition Table

Note that during training elements of a new input vector may be linked to both, either or neither output node. This represents the fact that a tuple can appear (i) in both a correct and incorrect string, (ii) in either or (iii) in neither. Tables 3 and 4 gives some information on the distribution of elements in training and testing sets. The data is asymmetric: any node that appears in a grammatical string can also appear in an ungrammatical one, but the reverse is not true. When training is completed links from unused inputs are enabled and their weights set to 0.0

Any single entry in the preliminary Prohibition Table (Section 5.4) can be omitted, and the pair or triple be included in the neural processing task. In this case a tuple that cannot occur in a grammatical string will, for Hodyne, only be connected to the “no” output node. Conversely, if we examine the linkage of the Hodyne net, those tuples that are only connected to the “no” output are candidates for inclusion in a constraint based rule. Of course there is a chance that a rare grammatical occurrence may show up as the size of the training set increases.

### 7.5 LMS network

The LMS model is based on the traditional method, described in “Parallel Distributed Processing” [37, page 322]. A bipolar activation function is used, and outputs are in the range  $-1$  to  $+1$ . As with the Perceptron, the output represents the  $\Gamma$  measure for the string being processed. Gradient descent is used to reduce the error between desired and actual outputs.

It has been known for many years in the numerical optimization field that the gradient descent technique is a poor, slow method [39]. This is now also accepted wisdom in the neural network community [28]. Other training methods, such as conjugate gradients, are usually preferable. For

this experiment, however, the traditional method has been used, but some variations to speed up training and improve performance have been incorporated. Brady et al. [40] described some anomalies that can arise with the traditional LMS model. As a remedy Sontag’s technique for interpreting the error measure is included [41]. This means that an error is only recorded if a vector falls into the wrong class. The target output is a threshold, and if this threshold is passed the vector is considered correctly classified. This contrasts with the original LMS method, in which an error is recorded if the target is either undershot or overshoot.

## 8 Performance

### 8.1 Training times

Since we have been developing a prototype the training threshold was taken so that training was fast - less than 10 seconds for the Perceptron, less than 20 seconds for the Hodyne network. Subject to this constraint the percentage of strings that could be trained ranged from 96.5% to 99.0%. The Perceptron was fastest. For the LMS net training times were between 1 and 2 orders of magnitude greater, but the inefficient gradient descent method was used.

### 8.2 Ability to generalize

Results are given in Tables 6 to 8. This system has been developed to produce a winning string for each sentence, and performance can be assessed on different measures of correctness, as described in Section 5.9. For the purpose of investigating the function of the networks we take the strictest measure, *correct-d* in the tables, requiring that strings should be classified correctly. However, we can interpret the results so that in practice we get up to 100% correct for our practical application, since a winning string may have a negative  $\Gamma$  measure. Thus the practical measure of correctness can be higher than the percentage of correctly classified strings.

Table 9 gives a summary of the results, showing how these vary with the ratio of test set size to training set size. This would be expected. If there is insufficient training data performance degrades sharply.

The Hodyne net performed well, and this architecture was used for the prototype. Previous work in this field compared the performance of multi-layer Perceptrons to that of single layer models, and found they performed less well. This is discussed in Section 9.

Training set	Test set	Pairs used	Triples used	Training threshold	correct-a %	correct-b %	correct-c %	correct-d %
Tr 1	Ts 1	Y	Y	95.0	100	97.6	95.2	92.9
		Y	Y	99.0	100	97.6	95.2	89.4
		Y		97.0	100	97.6	92.9	85.8
			Y	98.5	100	97.6	97.6	91.8
Tr 2	Ts 2	Y	Y	99.0	96.6	96.6	91.5	88.5
		Y		98.0	93.2	93.2	88.1	89.1
			Y	98.5	98.3	98.3	89.8	85.6
Tr 3	Ts 3	Y	Y	99.0	98.4	98.4	95.2	80.9
		Y		96.0	96.8	93.7	92.1	85.6
			Y	99.0	96.8	95.2	90.5	78.9
Tr 4	Ts 4	Y	Y	99.0	92.5	92.5	74.6	77.4
		Y		97.0	92.5	92.5	92.5	82.5
			Y	99.0	89.6	88.1	83.6	78.3

Table 6: Results using Perceptron. Recall that correct-a means hypertags are correctly placed, correct-b that words inside subject are correctly tagged also, correct-c that all words in part of sentence being processed are also correctly tagged, correct-d that the string is in the right class

Training set	Test set	Pairs used	Triples used	Training threshold	correct-a %	correct-b %	correct-c %	correct-d %
Tr 1	Ts 1	Y	Y	95.0	100	100	97.6	91.8
		Y	Y	99.0	100	100	100	92.9
		Y		97.0	100	97.6	95.2	90.6
			Y	98.5	100	100	100	89.4
Tr 2	Ts 2	Y	Y	99.0	100	100	94.9	91.4
		Y		98.0	100	100	94.9	90.2
			Y	98.5	98.3	98.3	94.9	96.8
Tr 3	Ts 3	Y	Y	99.0	100	98.4	95.2	89.2
		Y		96.0	98.4	98.4	95.2	91.2
			Y	99.0	96.8	96.8	92.1	86.1
Tr 4	Ts 4	Y	Y	99.0	95.5	94.0	91.0	84.4
		Y		98.0	95.5	94.0	92.5	83.0
			Y	99.0	94.0	94.0	83.6	82.5

Table 7: Results for Hodyne net on same training and testing data as for the Perceptron (Table 6)

Training set	Test set	Pairs used	Triples used	Training threshold	correct-c %	correct-d %
Tr 1	Ts 1	Y	Y	99.0	97.6	92.9
		Y		97.0	90.5	88.3
			Y	99.0	97.6	90.6
Tr 2	Ts 2	Y	Y	99.0	91.5	88.5
		Y		98.0	88.1	88.5
			Y	98.5	93.2	85.1
Tr 3	Ts 3	Y	Y	99.0	93.7	85.1
		Y		96.0	95.2	87.6
			Y	99.0	90.5	80.9
Tr 4	Ts 4	Y	Y	99.0	92.5	82.5
		Y		97.0	97.0	83.5
			Y	99.0	85.1	80.2

Table 8: Results for LMS net on the same data. Compare with Tables 6 and 7

Ratio test set / training set	Perceptron % test strings correct	Hodyne % test strings correct	LMS % test strings correct	Hodyne % hypertags correct
0.10	89.4	92.9	92.9	100
0.20	88.5	91.4	88.5	100
0.23	80.9	89.2	85.1	100
0.26	77.4	84.4	82.5	95.5

Table 9: Summary of results culled from Tables 2, 6, 7 and 8, showing performance on 4 different training and test sets.

## 9 Understanding the operation of the network

### 9.1 The importance of negative information

Consider the following unremarkable sentence, and some of the strings it generates:

the	determiner
directions	noun
given	past-part-verb
below	preposition or adverb
must	auxiliary verb
be	auxiliary verb
carefully	adverb
followed	past-part-verb
.	endpoint

string no. 1:  
strt [ det noun ] pastp adv aux aux adv pastp endp

string no. 2:  
strt [ det noun pastp ] prep aux aux adv pastp endp

string no. 3: \*\*\* target \*\*\*  
strt [ det noun pastp prep ] aux aux adv pastp endp

In the LOB corpus the pair (**preposition, modal-verb**), which represents the words (**below, must**)<sup>4</sup> has a frequency of less than 0.01%, if it occurs at all [42]. So when a sentence like this is processed in testing mode the particular construction may well not have occurred in any training string. However, in the candidate strings that are generated wrong placements should be associated with stronger negative weights *somewhere* in the string. For example, string 2 maps onto:

\* [ The directions given ] below must be carefully followed.

The proposed subject would not be associated with strong negative weights. However, the following pairs and triples include at least one that is strongly negative, such as ( ], **preposition**), an element in the negative strings generated in the training set. The correct placement, as in string 3, would be the least bad, the one with the highest  $\Gamma$  score.

By training on negative as well as positive examples we increase the likelihood that in testing mode a previously unseen structure can be correctly processed. In this way the probability of correctly processing rare constructions is increased.

### 9.2 Relationship between frequency of occurrence and weight

After training we see that the distribution of weights in Hodyne and Perceptron nets have certain characteristics in common. In both cases there is a trend for links on the least common input tuples to be more heavily weighted than the more common: see Figures 8 and 9.

This characteristic distribution of weights can be understood when we examine the process by which the weights are adapted. Since we are processing negative as well as positive examples in the training stage, the movement of weights differs from that found with positive probabilities

---

<sup>4</sup>Modal verbs are included in the class of auxiliary verb in this tagset

alone. Some very common tuples will appear frequently in both correct and incorrect strings. Consider a pair such as (**start-of-sentence**, **open-subject**). This will often occur at the start of both grammatical and ungrammatical strings. The result of the learning process is to push down the weights on the links to both the “yes” and the “no” output nodes.

A significant number of nodes represent those tuples that have never occurred in a grammatical strings. A few nodes represent tuples that have only occurred in grammatical strings (Tables 3 and 4).

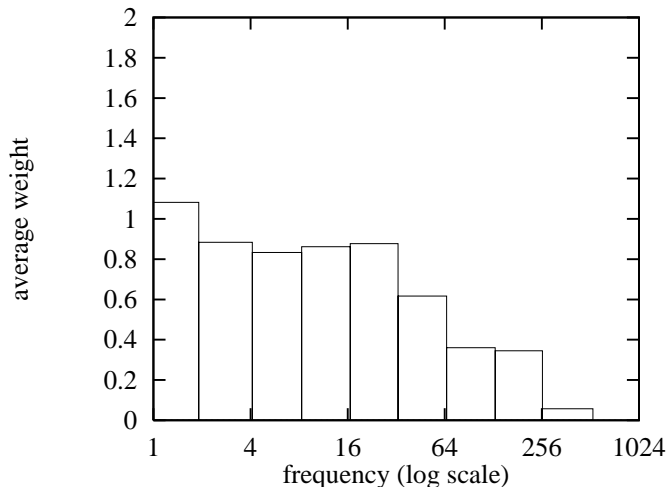


Figure 8: Weights plotted against frequency of input node occurring, for training corpus Tr 1 on Hodyne

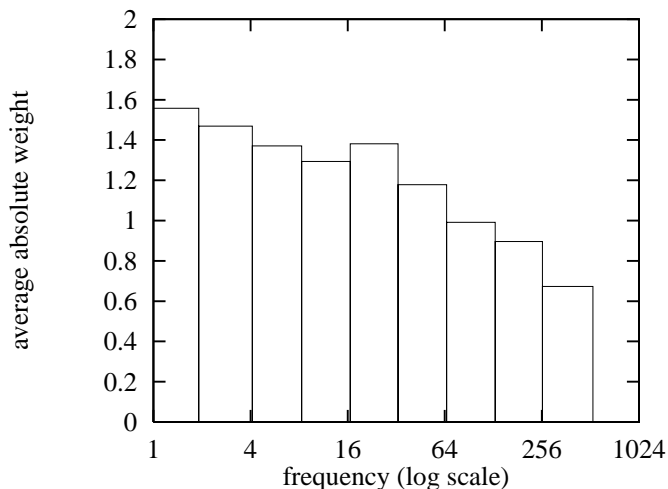


Figure 9: Weights plotted against frequency of input node occurring, for training corpus Tr 1 on Perceptron

The relationship between frequency of occurrence and level of weight accompanies the decision to assess the correctness of a whole string, rather than the status of each element. Strings that are slightly wrong will include tuples that occur both in grammatical and ungrammatical sequences. As a consequence we see that the classification decision can depend more on infrequently occurring

tuples. In particular, tuples that usually only occur in an ungrammatical string can have a significant influence on the classification task.

### 9.3 Direct update *versus* error minimization

The use of direct update rather than error minimization methods may also have an effect on generalization. The traditional LMS measure can lead to situations where most input vectors are close to the target, while a few, or a single one, are distant. This may be desirable when noisy data is processed, but not for our linguistic data, where we want precise fitting. We want to capture information from isolated examples. We want to classify strings that are ungrammatical in a single element as well as those that are grossly ungrammatical.

## 10 Conclusion

The original objective of this work was to see whether the pattern matching capabilities of neural networks could be mobilised for Natural Language Processing tasks. The working partial parser demonstrates that they can be.

Multi-layer Perceptrons were tried in the past, but it was found that single layer networks were more effective, provided that the data was appropriately converted to a higher order form. Some arguments against this approach centre on the lack of a principled method to find the pre-processing,  $\Phi$  function. But though the methods of finding the initial  $\Phi$  function are based on intuition, a close initial examination of the data can mean that this intuition is founded on an understanding of the data characteristics. In the case of the linguistic data support for the non-linear conversion function has come from information theoretic tools. Though the setting of parameters is not data driven at the micro level, as in a supervised learning environment, the functions are chosen to capture some of the structure and invariances of the data. The development of neural processors for an analysis of DNA sequences also illustrates this [22].

The analysis in the previous section illustrates the transparency of single layer networks, and indicates why they are such convenient tools. Compared to multi-layer Perceptrons, the parameters of the processor are more amenable to being interpreted. The Hodyne net in particular lends itself to further linguistic analysis. Furthermore, this approach has the advantage of fast two stage training. The speed of training, measured in seconds, shows how quickly single layer networks can fix their weights. Training times are hardly an issue.

A significant question of generalization ability is seen to relate to the ratio of testing to training data set size. Current work on generalization has focused on principled methods of determining training set size to ensure that the probability of generalization error is less than a given bound. Having implemented a preliminary prototype our work will continue with much larger corpora.

In the development of this technology we return to the fundamental question: how do we reconcile computational feasibility with empirical relevance? How do we match what can be done to what needs to be done? Firstly, in addressing the parsing problem we start by decomposing the problem into computationally more tractable subtasks. Then we investigate the data and devise a representation that enables the simplest effective processors to be used. The guiding principles are to attack complexity by decomposing the problem, and to adopt a reductionist approach in designing the neural processors.

## References

- [1] C Lyon and R Dickerson. A fast partial parse of natural language sentences using a connectionist method. In *7th Conference of the European Chapter of the Association of Computational Linguistics*, pages 215–222, 1995.
- [2] C Lyon and R Frank. Neural network design for a natural language parser. In *International Conference on Artificial Neural Networks (ICANN)*, pages 105–110, 1995.
- [3] S Holden and P Rayner. Generalization and PAC learning: Some new results for the class of generalized single layer networks. *IEEE Trans. on Neural Networks*, pages 368–380, 1995. (PAC is Probably Approximately Correct).
- [4] P J Wyard and C Nightingale. A single layer higher order neural net and its application to context free grammar recognition. In R Linggard, D J Myers, and C Nightingale, editors, *Neural Networks for Vision, Speech and Natural Language*, pages 203–234. Chapman and Hall, 1992.
- [5] H Cunningham, R J Gaizauskas, and Y Wilks. A general architecture for text engineering (GATE). Technical report, Institute for Language, Speech and Hearing (ILASH), University of Sheffield, 1996.
- [6] E M Gold. Language identification in the limit. *Information and Control*, 10:447–474, 1967.
- [7] E Charniak. *Statistical Language Learning*. MIT Press, 1993.
- [8] E Brill et al. Deducing linguistic structure from the statistics of large corpora. In *DARPA Speech and Natural Language Workshop*, pages 275–281, 1990.
- [9] C E Shannon. Prediction and Entropy of Printed English. *Bell System Technical Journal*, pages 50–64, 1951.
- [10] T Dunning. Accurate methods for the statistics of surprise and coincidence. *Computational Linguistics*, pages 61–73, 1993.
- [11] E Atwell. Constituent-likelihood grammar. In R Garside, G Leech, and G Sampson, editors, *The Computational Analysis of English: a corpus-based approach*, pages 57–65. Longman, 1987.
- [12] C Lyon. *The representation of natural language to enable neural networks to detect syntactic structures*. PhD thesis, University of Hertfordshire, 1994.
- [13] T Sejnowski and C Rosenberg. Parallel networks that learn to pronounce English text. *Complex Systems*, pages 145–168, 1987.
- [14] J L Elman. Distributed representations, simple recurrent networks and grammatical structure. *Machine Learning*, pages 195–223, 1991.
- [15] C L Giles et al. Higher order recurrent networks and grammatical inference. In D S Touretzky, editor, *Advances in neural information processing systems*, pages 380–388. Morgan Kaufmann, 1990.
- [16] C L Giles et al. Learning and extracting finite state automata with second order recurrent neural networks. *Neural Computation*, pages 393–405, 1992.
- [17] T M Cover and J A Thomas. *Elements of Information Theory*. John Wiley and Sons Inc., 1991.
- [18] R Garside. The CLAWS word-tagging system. In R Garside, G Leech, and G Sampson, editors, *The Computational Analysis of English: a corpus based approach*, pages 30–41. Longman, 1987.
- [19] P. Pym. Perkins engines and publications. In *PROCEEDINGS of Technology and Language in Europe 2000*. DGXIII-E of the European Commission, 1993.
- [20] Yoh-Han Pao. *Adaptive pattern recognition and neural networks*. Addison Wesley, 1989.
- [21] B Widrow and M Lehr. 30 years of adaptive neural networks. *Proceedings of IEEE*, 78:1415–1442, September 1990.
- [22] A Lapedes, C Barnes, C Burks, R Farber, and K Sirotkin. Applications of neural networks and other machine learning algorithms to DNA sequence analysis. In *Computers and DNA*, pages 157–182. Addison Wesley, 1992.

- [23] R Duda and P Hart. *Pattern Classification and Scene Analysis*. John Wiley, 1973.
- [24] S Holden and M Niranjan. On the practical applicability of VC dimension bounds. *Neural Computation*, 7(6):1265–1288, 1995.
- [25] A K Chhabra, S Chandran, and R Kasturi. Table structure interpretation and neural network based text recognition for conversion of telephone company tabular drawings. In *International Workshop on Applications of Neural Networks to Telecommunications*, pages 92–98, Princeton, 1993.
- [26] S Roberts and L Tarassenko. A new method of automated sleep quantification. *Medical and Biological Engineering and Computing*, pages 509–517, September 1992.
- [27] Z Wang, Z Ulanowski, and P H Kaye. On solving the inverse scattering problem with RBF neural networks. Technical report, ERDC, University of Hertfordshire, 1996.
- [28] C M Bishop. *Neural Networks for Pattern Recognition*. OUP, 1995.
- [29] M Minsky and S Papert. *Perceptrons*. MIT Press, 3rd edition, 1988. Epilog added to 1969 edition.
- [30] J Shavlik, R Mooney, and G Towell. Symbolic and neural learning algorithms: An experimental comparison. *Machine Learning*, pages 111–140, 1991.
- [31] J Sklansky and G Wassel. *Pattern Classifiers and Trainable Machines*. Springer-Verlag, 1981.
- [32] J-H Lin and J Vitter. Complexity results on learning by neural nets. *Machine Learning*, pages 211–230, 1991.
- [33] T C Bell, J G Cleary, and I H Witten. *Text Compression*. Prentice Hall, 1990.
- [34] T M Cover. Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE Transactions on Electronic Computing*, 14:326–334, 1965.
- [35] J Hertz, A Krogh, and R Palmer. *Introduction to the Theory of Neural Computation*. Addison Wesley, 1991.
- [36] E Baum and D Haussler. What size net gives valid generalization? *Neural Computation*, 1:151–160, 1989.
- [37] D Rumelhart and J McClelland. *Parallel Distributed Processing*. MIT, 1986.
- [38] C Lyon. Using single layer networks for discrete, sequential data: an example from natural language processing (extended version). Technical report, School of Information Sciences, University of Hertfordshire, April 1996.
- [39] L C W Dixon. *Nonlinear Optimisation*. English Universities Press Ltd., 1972.
- [40] M L Brady, R Raghavan, and J Slawny. Back propagation fails to separate where perceptrons succeed. *IEEE Trans. on Circuits and Systems*, pages 665–674, 1989.
- [41] E D Sontag and H J Sussmann. Back propagation separates where perceptrons do. *Neural Networks*, 4:243–249, 1991.
- [42] S. Johansson and K. Hofland. *Frequency analysis of English vocabulary and grammar*. Clarendon, 1989.